digital whisper
Technologic Papers

על לבבות שבורים - CVE-2014-0160 / Heartbleed

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

החודש, שלושה חוקרי אבטחת מידע (Matti Kamunen, Antti Karjalainen ו-Riku Hietamäki) מחברת Codenomicon פרסמו תוצאות מחקר שהם ביצעו על חולשה שהתגלתה ע"י Neel Mehta מגוגל, בספריה OpenSSL. מאז פרסום החולשה אין כמעט בלוג או אתר חדשות אחד בתחום שלא כתב עליה. מדובר באחת הספריות הנפוצות ביותר (אם לא הנפוצה ביותר) למימוש SSL/TLS בשירותי WEB (ומעבר לזאת - מספר רב של מוצרי אבטחה שעושים שימוש בהצפנת התעבורה שלהם עושים שימוש בספריה הנ"ל, כגון נתבים של Cisco, Juniper וכו').

אבל לפני הכל, שני דברים שחשוב לדעת:

- בהרבה מקומות שראיתי יש הרבה מאוד בלבול, שלא כמו במתקפות קודמות (כגון Lucky13CRIME, BREACH, וכו') שהתגלו ב-SSL/TLS, כשל האבטחה במקרה של Heartbleed הוא אינו בפרוטוקול עצמו אלא במימוש שלו ב-OpenSSL.
- הבאג (כשלעצמו) לא מאפשר להשתלט על השרת שמריץ את גרסאת ה-OpenSSL הפגיעה.

אחרי שהפנמנו את שני הנקודות הנ"ל, אפשר להתחיל.

קצת על OpenSSL

OpenSSL הינו פרוייקט קוד-פתוח שנועד לאפשר מימוש חופשי של הפרוטוקולים SSLv2, SSLv3 ו-TLSv1. הפרוייקט פורסם לראשונה בסוף שנת 1998 והוא מבוסס על פרוייקט ישן יותר, בשם SSLeay שפותח במקור ע"י Eric Andrew Young ו-Tim Hudson. גרסאתו האחרונה של OpenSSL הינה 1.0.2.

אז מה זה Heartbleed?

לפני ששואלים את השאלה הנ"ל, יש צורך לשאול קודם לכן, את השאלה "מה זה Heartbeat" אך בכלליות: Heartbleed היא חולשה שהתגלתה במנגנון ה-Heartbeat, עליו אסביר בשורות הבאות. החולשה קיימת ב-OpenSSL מגרסאות 1.0.1 עד 1.0.1f (כולל). גרסאות מעל (1.0.1g) וגרסאות מתחת (1.0.0 ומטה) אינן פגיעות.

אז מה זה Heardbeat?

כפי שמספק לנו ה-RFC6520 (TLS/DTLS Heartbeat Extension), הקמת חיבור TLS (ובכלליות, הקמה של חיבורים מוצפנים) דורשים לא מעט משאבים ולוקחים לא מעט זמן (כמובן, ביחס להקמה של חיבורים שאינם מאובטחים), מפני שבהרבה מקרים הם דורשים לבצע מספר חישובים מתמטיים. הרעיון של Heardbeat נועד לחסוך את הצורך בהקמה של חיבור כזה כל מספר שניות - הרעיון הוא שלאחר יצירת חיבור ראשוני ה-Client יוכל להחזיק את החיבור "חי" גם אם **כרגע** אין לו צורך לשלוח בו לשליחת מידע ע"י שליחת חבילות TLS1_HB_REQUEST.

הגדרת התפקיד ב-RFC:

> 5.2. Liveliness Check:
> Sending HeartbeatRequest messages allows the sender to make sure that it can reach the peer and the
> peer is alive. Even in the case of TLS/TCP, this allows a check at a much higher rate than the TCP keep-
> alive feature would allow.

> למען הידע הכללי: לחבילות ה-Heartbeat יש שימוש נוסף, אך פחות נפוץ, כאשר רוצים לבצעPath MTU Discovery במקרים שבהם לא רוצים להשתמש בפרוטוקולי Stream Control חיצוניים.

אם נסתכל ב-RFC (עמוד 4) נראה שהמבנה של חבילת Heartbeat כזאת נראה כך:

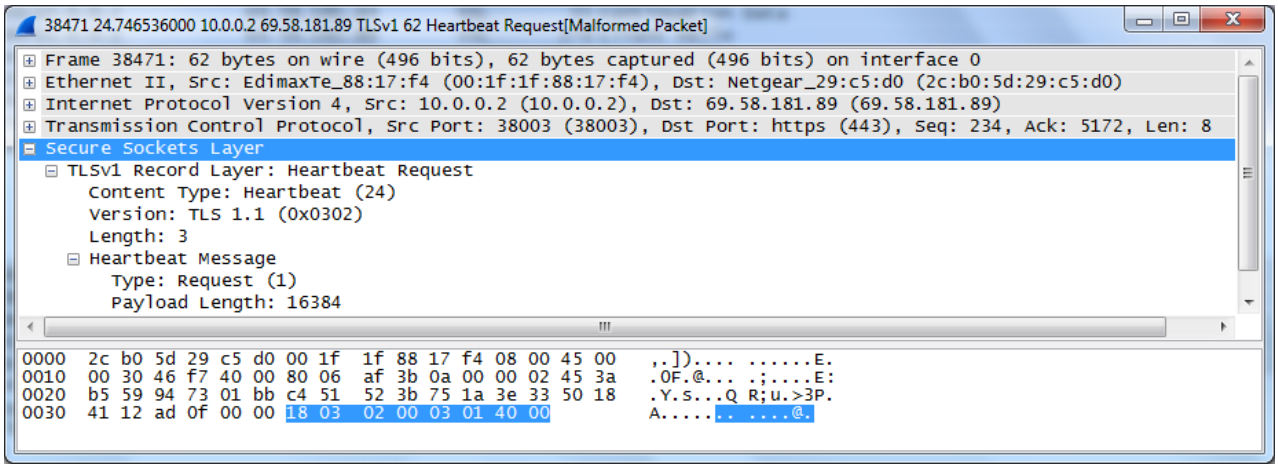
```
struct {
      HeartbeatMessageType type;
      uint16 payload_length;
      opaque payload[HeartbeatMessage.payload_length];
      opaque padding[padding_length];
} HeartbeatMessage;
```

השדה הראשון (**type**) מגדיר את סוג חבילת ה-Heartbeat (החבילה יכולה להיות TLS1_HB_REQUEST או TLS1_HB_RESPONSE), ובשני המקרים, היא לא יכולה להיות גדולה להיות יותר מ-2^14 בתים או כפי שהוגדר ב-max_fragment_length בזמן הקמת התקשורת (פרטים נוספים על שלב זה ב-RFC6066).

השדה השני (**payload_length**) מגדיר את גודל ה-Payload שנשלח עם החבילה, והשדה השלישי (**payload**) מכיל את תוכן החבילה עצמה.

השדה הרביעי (**padding**) מכיל מידע אקראי לצורך ריפוד בגודל: TLSPlaintext.length - payload_length - 3 (גודל החבילה, פחות ה-payload_length פחות 3, בגלל שגודל השדה type הוא בית אחד וגודל שדה ה-payload_length הוא שני בתים).

דוגמא לחבילה כזאת:



[את החבילה שלחתי ל-www.verisign.com , באמת רק לשם הבדיחה...]

לפי התמונה, ניתן לראות כי מבנה החבילה מורכב משדה שקובע ראשית את סוג החבילה, לאחר מכן שדה שמורה על גודל ה-payload, ולאחר מכן ה-payload עצמו.

כאמור, שרת מזהה כי חבילת ה-Heartbeat הינה TLS1_HB_REQUEST ע"י כך ששדה ה-Type שווה ל-1. במקרים כאלה, על מנת לשמור על קשר עם הלקוח, עליו להגיב עם חבילת TLS1_HB_RESPONSE מתאימה. אם נסתכל בקוד של OpenSSL, תחת הפונקציה הפגיעה (הפונקציה tls1_process_heartbeat בעת הטיפול בחבילות TLS, ו-dtls1_process_heartbeat בעת הטיפול ב-DTLS (תחת t1_lib.c ו-d1_lib.c, בהתאמה) שאחראית לטפל בחבילות Heartbeat , נראה את הקוד הבא:

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
...
if (hbtype == TLS1_HB_REQUEST)
```

```
{
   unsigned char *buffer, *bp;
   int r;

   /* Allocate memory for the response, size is 1 byte
    * message type, plus 2 bytes payload length, plus
    * payload, plus padding
    */
   buffer = OPENSSL_malloc(1 + 2 + payload + padding);
   bp = buffer;
   /* Enter response type, length and copy payload */
   *bp++ = TLS1_HB_RESPONSE;
   s2n(payload, bp);
   memcpy(bp, pl, payload);
   /* Random padding */
   RAND_pseudo_bytes(p, padding);

   r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload +
padding);
   …
}
else if (hbtype == TLS1_HB_RESPONSE)
{
   unsigned int seq;

   /* We only send sequence numbers (2 bytes unsigned int),
    * and 16 random bytes, so we just try to read the
    * sequence number
    */
   n2s(pl, seq);

   if (payload == 18 && seq == s->tlsext_hb_seq)
   {
      dtls1_stop_timer(s);
      s->tlsext_hb_seq++;
      s->tlsext_hb_pending = 0;
   }
}
```

בשלב הראשון ניתן לראות כי הפונקציה מקבלת שני פרמטרים **שהגיעו מהמשתמש** עם חבילת ה-TLS1_HB_REQUEST). הפרמטרים הם סוג החבילה (הבית הראשון של payload מוכנס למשתנה hbtype) ואורך ה-payload (הפונקציה n2s() מכניסה את שני הבתים הבאים ב-payload ל-p) שהמשתמש שלח, ולאחר מכן, המשתנה pl יצביע על מיקום תחילת ההודעה ב-payload.

לאחר מכן, ניתן לראות כי קטע הקוד מחולק לשני חלקים עיקריים: הטיפול בחבילה מסוג TLS1_HB_REQUEST והטיפול בחבילה מסוג TLS1_HB_RESPONSE. מה שמעניין אותנו בשלב זה זה הקוד שאחראי על חבילות ה-TLS1_HB_REQUEST.

לאחר הבדיקה כי אכן מדובר בחבילת request, מתבצע הקוד הבא: הפונקציה OPENSSL_malloc() (המקבילה של malloc()), ובעזרתה מאלקצים ל-buffer מקום בזיכרון בגודל:

```
1 + 2 + payload + padding
```

וזאת כאמור, בגלל שגודל החבילה המוחזרת הינה גודל ה-payload, גודל ה-padding שיש להוסיף, ושני ערכים - הראשון מורה על סוג החבילה (שוקל בית אחד), והשני הוא מספר המורה על גודל עצמו (שוקל שני בייתים). בהמשך, המצביע bp מצביע אל המיקום בזיכרון של buffer, וזאת לטובת הכנת חבילת המידע שעל השרת להחזיר למשתמש.

כעת, מתבצעת הרכבת החבילה שעל השרת לשלוח כתגובה למשתמש:

```
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

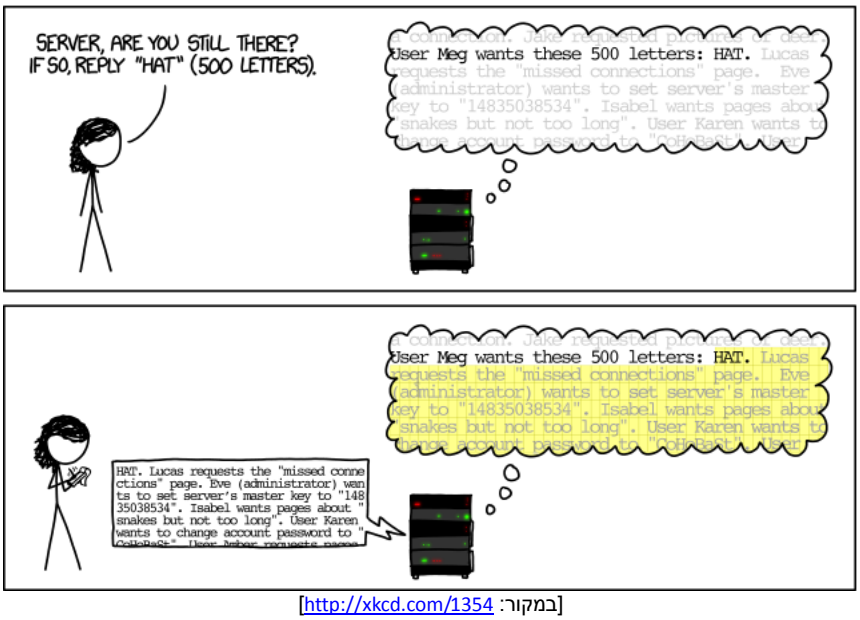
במיקום הראשון בחבילה מוכנס סוג החבילה (TLS1_HB_RESPONSE) - כמו שראינו ב-PCAP, השדה הראשון בחבילה, שוקל בית אחד ומורה על סוגה.

לאחר מכן, בעזרת הפונקציה s2n(), מועברים 2 הבתים הבאים ב-payload ומועברים ל-bp. ובסופו של דבר, בעזרת memcpy() מעתיקים ל-bp את payload (כמות) התווים מ-pl עצמו. לאחר הרכבת החבילה, השרת מוסיף padding לחבילה על פי הצורך והחבילה נשלחת ליעדה.

אז איפה בעצם החולשה?

מופיעה בדיוק כמה שורות מעל הכותרת ☺, בשלב בו memcpy() מעתיקה את המידע מ-pl אל pb. אם נסתכל בקוד, נראה כי השימוש ב-memcpy() מתבצע על מנת להעביר payload בתים מ-pl אל pb, ואם נסתכל בתחילת הקוד, נראה כי את הערך הקיים ב-payload המשתמש קובע! מה זאת אומרת? זאת אומרת שלא מתבצעת שום בדיקה בצד השרת ואף אחד לא מבטיח לנו שגודל ה-payload שהמשתמש שלח בפועל, אכן תואם את הערך הקיים בשדה payload_length (שגם הוא נשלח מהמשתמש) שעל פיו אנו קובעים את גודל החבילה שתוחזר למשתמש. מכאן שהמשתמש יכול לדווח כי הוא שלח משהו בגודל אחד, אך בפועל לשלוח מידע בגודל שונה.

אז איך אפשר לנצל זאת לטובתנו? בקצרה? באופן הבא:



[במקור: http://xkcd.com/1354]

ולא בקצרה:

כאשר השרת מעוניין להגיב לחבילת ה-TLS1_HB_RESPONSE, עליו להרכיב את TLS1_HB_RESPONSE בעזרת מידע שהגיע מהמשתמש. אחד מהפרמטרים הנ"ל הוא גודל החבילה שעליו להחזיר - וזאת הוא קובע (כפי שראינו) על פי הערך הקיים ב-payload, וזאת, כמו שראינו - מגיע מהמשתמש מבלי לוודא כי אכן זהו גודל ה-payload המקורי שהמשתמש שלח.

אם נעבור על סנאריו "רגיל", נראה שהמהלך נראה כזה: המשתמש שולח לשרת מחרוזת בעלת **X תווים**, ומספר המסמל את אורך המחרוזת: **X**. השרת מקצה מקום בזיכרון (בערימה של התהליך) שלו (שומר מצביע לשם) ומכניס את המחרוזת לשם. כעת על השרת להחזיר את אותה המחרוזת על מנת להגיב לחבילת ה-Heartbeat בצורה תקינה. הוא ניגש לאותו איזור בזיכרון, ושולף מהתא אליו מצביע המצביע למחרוזת **X תווים** - ומחזיר אותם למשתמש. המשתמש מקבל את המחרוזת - משווה למחרוזת שהוא שלח - ומבין כי הסשן עדיין חי.

אך אם נעבור על סנאריו "זדוני", נראה שהמהלך יראה כך: תוקף שולח לשרת מחרוזת בעלת **X תווים**, **ומספר הגדול בהרבה מאורך המחרוזת**: **Y**. ושוב, השרת מקצה מקום בערימה (ושומר מצביע לשם) ומכניס את המחרוזת לשם. כעת, על השרת להחזיר את אותה המחרוזת. הוא ניגש לאותו איזור בזיכרון שאליו מצביע המצביע ושולף **Y** תווים. **Y התווים הנ"ל כוללים את המחרוזת שהתוקף שלח ועוד Y-X תווים אקראיים הנמצאים בהמשך הזיכרון בערימה של** התהליך. השרת שולח את אותם תווים לתוקף, ומכאן - GAME OVER.

חשוב לזכור שאין לנו שליטה על מה שהשרת יחזיר, ולכן בהרבה מקרים מה שנקבל לאו דווקא יכלול מידע מעניין, אך לתוקף אין הגבלה על כמות הפעמים שהוא יוכל לנצל את החולשה הנ"ל. המגבלה היחידה שיש לתוקף היא גודל החבילה עליו הוא מדווח (Y), מפני שכמו שראינו בתחילה, על פי ה-RFC6520, שדה זה מוגבל ל-2^14kb (פחות 3 בתים שעליו לשלם בעת יצירת החבילה), ובכל הרצה התוקף יכול לקבל במקסימום Chunk של 64kb מהערימה של התהליך.

הקמת מעבדה

מצרכים:

- שרת HTTP שיודע להשתמש ב-OpenSSL (לדוגמא: Apache).
- גרסאת OpenSSL פגיעה (כאמור, כל גרסאת OpenSSL מ-1.0.1a עד 1.0.1f).
- Python
- Wireshark

על מנת להקים סביבה שבה נוכל להתנסות בנושא הנ"ל, בחרתי להשתמש בשרת Apache 2.2.21 (שמגיע עם XAMPP 1.7.7, אבל זה לא באמת משנה, פשוט זה מה שהיה אצלי מותקן), ניתן להוריד XAMPP מכאן:

https://www.apachefriends.org/download.html

אחרי שהתקנו את השרת, עלינו ליצור תעודת SSL, נעשה זאת באופן הבא:

- פתיחת Cmd וניווט לתיקיה בה התקנו את השרת. ניווט לתיקיה bin, שם לאתר את openssl ולהריצו באופן הבא:

```
openssl
```

- כעת אנו ב-Cli של OpenSSL, נבקש ליצור מפתח פרטי באורך 1024 ביט, לטובת יצירת התעודה, נעשה זאת כך:

```
OpenSSL> genrsa -des3 -out server.privkey 1024
```

- נתבקש להזין (פעמים) מחרוזת שתהווה passphrase ליצירת המפתח הפרטי של השרת. הרעיון בשימוש במפתח הפרטי שיצרנו, הוא שהשרת יוכל להשתמש בתעודה מבלי הצורך שנכניס את הסיסמה שלנו בכל פעם שנרצה להעלות אותו.

- לאחר מכן, יווצר לנו קובץ בשם server.privkey, שיכלול את המפתח הפרטי, אנו נשתמש בו על מנת לבצע את שלב ה-CSR (קיצור של Certificate Signing Request). ב-Cli של OpenSSL נכתוב:

```
OpenSSL> req -new -key server.privkey -config "..\..\php\extras\openssl\openssl.cnf" -out
server.csr
```

- כעת נתבקש להזין את פרטי התעודה (עיר, מדינה, שם הארגון, שם פרטי וכו', בשלב זה אנו יוצרים Self-Signed Certificate, ככה שהפרטים הללו לא באמת צריכים להיות אמיתיים).

- לאחר סוף השלב הקודם, נוצר לנו קובץ בשם server.csr שמהווה בעצם את הבקשה שלנו ליצירת SSL Certificate, כעת נוכל לגשת ליצירת התעודה, אך לפני כן - חשוב שנסיר את ה-passphrase מקובץ ה-server.privkey.

נעשה זאת כך:

```
OpenSSL> rsa -in server.privkey -out server.key
```

בשלב זה נתבקש נתבקש להכניס את ה-passphrase ל-server.privkey.

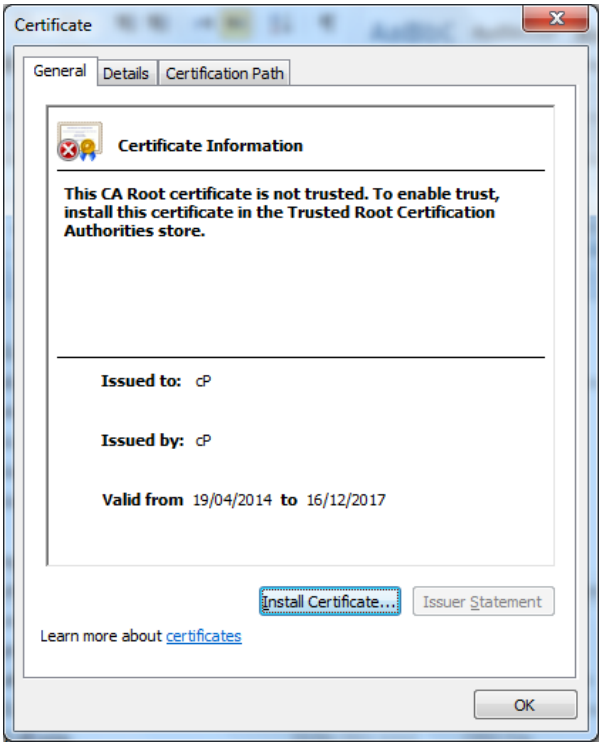
כעת ניצור את התעודה, באופן הבא:

```
OpenSSL> openssl x509 -req -days 1337 -in server.csr -signkey server.key -out server.crt
```

- במידה והכל עבד כשורה, נקבל פלט בסיגנון הבא:

```
Signature ok
subject=/C=IL/ST=TA/L=Bla/O=DigitalWhisper/OU=DigitalWhisper/CN=cP/emailAddress=xxx@xx
x.com
Getting Private key
```

ואם נסתכל בתיקיה, נראה שיש לנו קובץ בשם server.crt - שבעצם מהווה את תעודת ה-SSL שלנו:



על מנת שנוכל להשתמש בחתימה הנ"ל בשרת ה-Apache, נעביר אותה לתיקיה:

```
\apache\conf\ssl.crt\
```

ואת המפתח הפרטי של השרת, נעביר לתיקיה:

```
\apache\conf\ssl.key\
```

בשלב זה, עלינו להגדיר את Apache כך שידע לתמוך בחיבורי SSL בפורט 443 (או כל פורט אחר שנבחר), נעשה זאת ע"י:

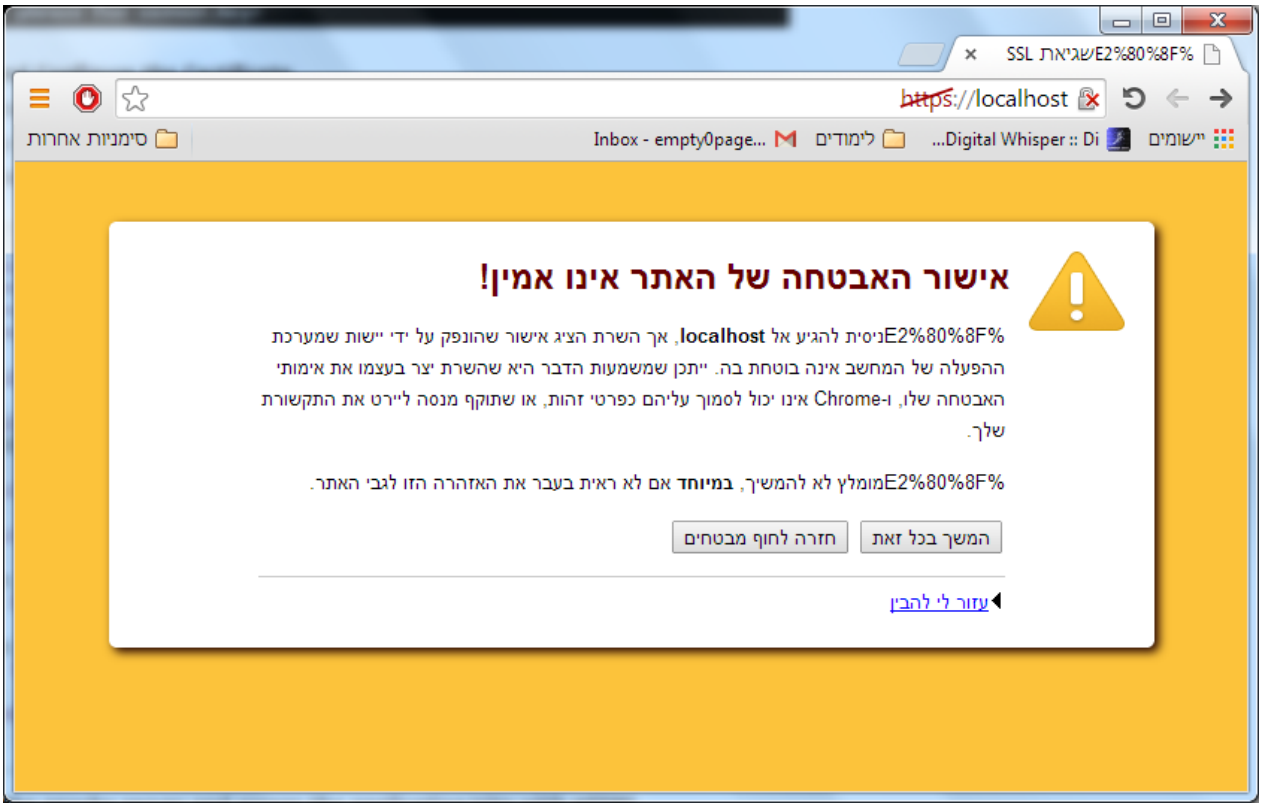
- הורדת ה-# בקובץ \apache\conf\httpd.conf מהשורה הבאה:

```
#Include "conf/extra/httpd-ssl.conf"
```

- הורדת ה-# בקובץ \apache\conf\extra\httpd-ssl.conf מהשורות הבאות:

```
Listen 443
DocumentRoot "C:/xampp/htdocs"
ServerName localhost:443
```

- וזהו, כל שנותר הוא לבצע Restart לשרת ה-Apache ולגלוש ל-https://localhost:



השגיאה הנ"ל נוצרת אך ורק בגלל שמי שחתם על התעודה (אנחנו) הוא לא גוף מורשה, וזה בסדר גמור - מה שמעניין אותנו זה רק שיהיה שרת HTTPS על המחשב לצורך המעבדה.

כעת, יש "בעיה" נוספת, והיא ששרת ב-Apache שמגיע עם XAMPP מגיע עם **OpenSSL 1.0.0e**:

Apache Version	Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
Apache API Version	20051115
Server Administrator	webmaster@localhost
Hostname:Port	localhost:443
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100

שהיא **אינה פגיעה ל-Heartbleed**, ולכן אנו נדרשים להוריד גרסאת OpenSSL פגיעה ולהחליף אותה במקורית. ניתן להוריד גרסאות OpenSSL פגיעות מהקישור הבא:

http://indy.fulgan.com/SSL/

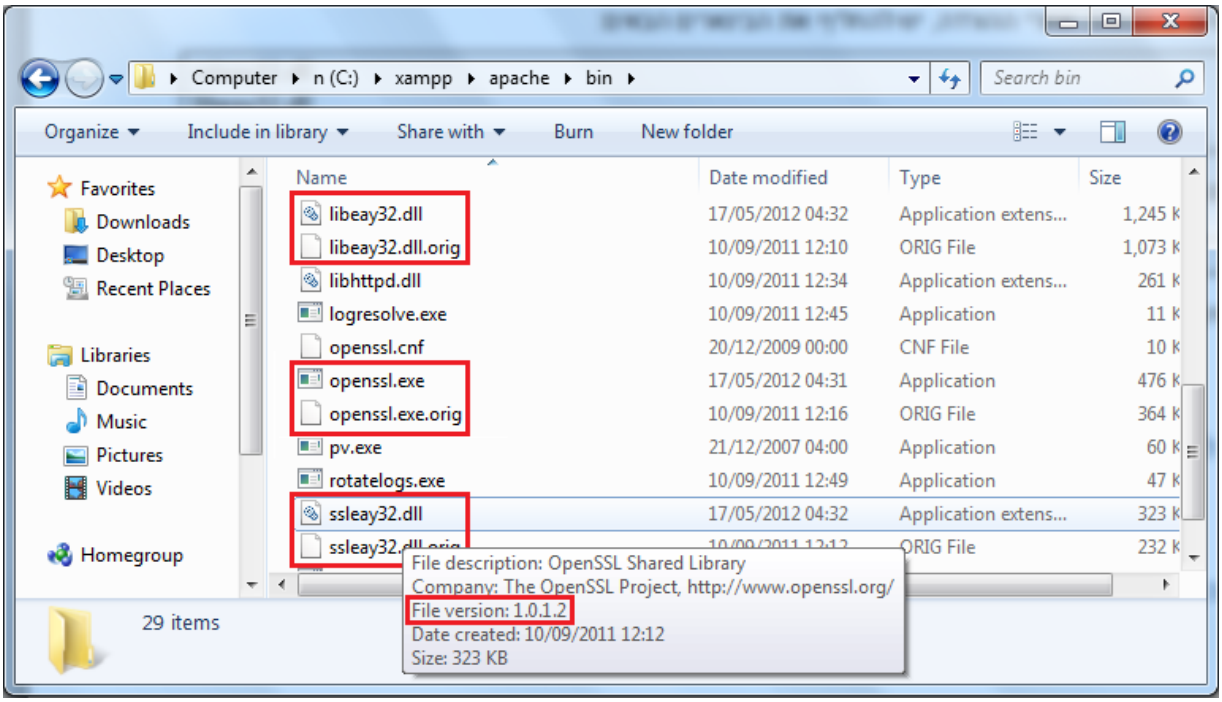
אני בחרתי ב-1.0.1b. אחרי ההורדה, יש להחליף את הבינארים הבאים:

```
ssleay32.dll
libeay32.dll
openssl.exe
```

בקבצים הנמצאים במיקום:

```
\apache\bin\
```

כמו בתמונה הבאה:



[אפשר לראות את הגרסא:1.0.1.2 - גרסא פגיעה של התוכנה]

וזהו, נשאר רק לבצע Restart לשרת, וכעת גרסאת ה-OpenSSL הפגיעה - טעונה:

Apache Version	Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.1b PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
Apache API Version	20051115
Server Administrator	webmaster@localhost
Hostname:Port	localhost:443
Max Requests	Per Child: 0 - Keep Alive: on - Max Per Connection: 100

אז עכשיו יש לנו שרת פגיע שרק מחכה לנו, נוכל להשתמש בכל אחד מהכלים בפרק הבא על מנת לשלוף ממנו מידע באמצעות החולשה.

כלים לאיתור שרתים פגיעים

כמו בכל חשיפת חולשה בסדר גודל שכזה, נכתבים הרבה מאוד כלים שנועדו לעזור לנו לבדוק האם השרת פגיע, בסופו של דבר, רב הסקריפטים שתמצאו - יעבדו באותה הדרך: יצירת חיבור TLS עם הכתובת שניתן לה, שליחת חבילת Heartbeat בגודל X ובדיקה האם קיבלנו תשובה הגדולה יותר מ-X. במידה וכן - נוכל לדעת כי השרת פגיע. במידה ולא - או שהשרת מריץ גרסא מתוקנת, או שהשרת מריץ גרסא שבכלל לא תומכת בחבילות Heartbeat.

דוגמאות לסקריפטים כאלה ניתן למצוא בקישורים הבאים:

- https://gist.github.com/sh1n0b1/10100394
- https://github.com/sensepost/heartbleed-poc
- https://github.com/musalbas/heartbleed-masstest/blob/master/ssltest.py
- https://access.redhat.com/labs/heartbleed/heartbleed-poc.py
- https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl_heartbleed.rb
- https://svn.nmap.org/nmap/scripts/ssl-heartbleed.nse

דוגמאות לאתרים המציעים בדיקות On-Line:

- https://filippo.io/Heartbleed/
- https://www.ssllabs.com/ssltest/index.html

נשתמש בסקריפט של nmap, שנכתב ע"י Patrik Karlsson ומבוסס ברובו על סקריפט פייתון של Jared Stafford, ניתן להוריד את הסקריפט מכאן:

- https://svn.nmap.org/nmap/scripts/ssl-heartbleed.nse

על מנת להשתמש בסקריפט הנ"ל יש להתקין nmap מאחת הגרסאות האחרונות (מעל 6.25, פשוט תעדכנו לאחרונה: 6.46) - מפני שהסקריפטים החדשים של nmap (כגון זה) דורשים גרסאת lua חדשה שלא תומכת לאחור, ניתן להוריד אותה מכאן:

- http://nmap.org/download.html

במידה ויש לכם גרסאת nmap שלא כוללת את tls.lua, ניתן להוריד את הקובץ מכאן:

- https://svn.nmap.org/nmap/nselib/tls.lua

ולאחר מכן, יש להוריד את קובץ ה-nse לתיקיה:

```
\Nmap\nselib\ssl-heartbleed.nse
```

כעת נוכל לבצע את הסריקה באופן הבא:

```
nmap -p 443 --script ssl-heartbleed -sV 10.0.0.2 --script-trace
```

אנחנו משתמשים ב-script-trace על מנת לראות את ה-dump של החבילות שהסקריפט שלח / קיבל, ככה שבמידה והשרת אכן פגיע - נוכל לראות את המידע שהוא החזיר.

דוגמא ל-Dump משרת ה-XAMPP שהרגע הקמנו:

```
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 2F 2A 3B 71  ....#......./*;q
00e0: 3D 30 2E 31 0D 0A 55 73 65 72 2D 41 67 65 6E 74  =0.1..User-Agent
00f0: 3A 20 4D 6F 7A 69 6C 6C 61 2F 35 2E 30 20 28 57  : Mozilla/5.0 (W
0100: 69 6E 64 6F 77 73 20 4E 54 20 36 2E 31 29 20 41  indows NT 6.1) A
0110: 70 70 6C 65 57 65 62 4B 69 74 2F 35 33 37 2E 33  ppleWebKit/537.3
0120: 36 20 28 4B 48 54 4D 4C 2C 20 6C 69 6B 65 20 47  6 (KHTML, like G
0130: 65 63 6B 6F 29 20 43 68 72 6F 6D 65 2F 33 34 2E  ecko) Chrome/34.
0140: 30 2E 31 38 34 37 2E 31 31 36 20 53 61 66 61 72  0.1847.116 Safar
0150: 69 2F 35 33 37 2E 33 36 0D 0A 52 65 66 65 72 65  i/537.36..Refere
0160: 72 3A 20 68 74 74 70 73 3A 2F 2F 6C 6F 63 61 6C  r: https://local
0170: 68 6F 73 74 2F 70 68 70 6D 79 61 64 6D 69 6E 2F  host/phpmyadmin/
0180: 69 6E 64 65 78 2E 70 68 70 3F 74 6F 6B 65 6E 3D  index.php?token=
0190: 34 35 61 31 62 38 33 39 35 31 31 34 30 32 32 64  45a1b8395114022d
01a0: 66 31 37 39 35 35 32 65 34 31 66 31 31 64 64 65  f179552e41f11dde
01b0: 0D 0A 41 63 63 65 70 74 2D 45 6E 63 6F 64 69 6E  ..Accept-Encodin
01c0: 67 3A 20 67 7A 69 70 2C 64 65 66 6C 61 74 65 2C  g: gzip,deflate,
01d0: 73 64 63 68 0D 0A 41 63 63 65 70 74 2D 4C 61 6E  sdch..Accept-Lan
01e0: 67 75 61 67 65 3A 20 68 65 2D 49 4C 2C 68 65 3B  guage: he-IL,he;
01f0: 71 3D 30 2E 38 2C 65 6E 2D 55 53 3B 71 3D 30 2E  q=0.8,en-US;q=0.
0200: 36 2C 65 6E 3B 71 3D 30 2E 34 0D 0A 43 6F 6F 6B  6,en;q=0.4..Cook
0210: 69 65 3A 20 70 6D 61 5F 6C 61 6E 67 3D 65 6E 3B  ie: pma_lang=en;
```

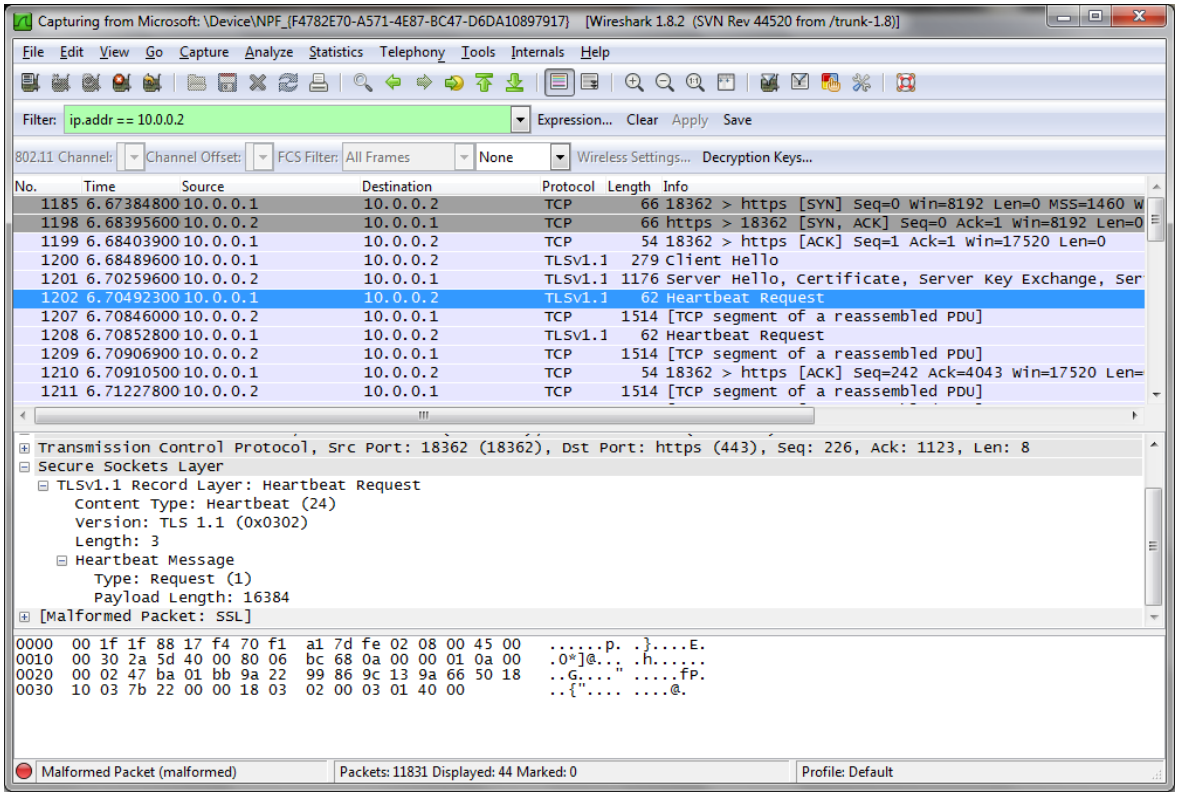
```
0220: 20 70 6D 61 5F 63 6F 6C 6C 61 74 69 6F 6E 5F 63   pma_collation_c
0230: 6F 6E 6E 65 63 74 69 6F 6E 3D 75 74 66 38 5F 67  onnection=utf8_g
0240: 65 6E 65 72 61 6C 5F 63 69 3B 20 70 6D 61 5F 6D  eneral_ci; pma_m
0250: 63 72 79 70 74 5F 69 76 3D 55 79 52 34 30 33 65  crypt_iv=UyR403e
0260: 7A 71 33 51 25 33 44 3B 20 70 68 70 4D 79 41 64  zq3Q%3D; phpMyAd
0270: 6D 69 6E 3D 6F 38 68 76 6C 30 74 68 30 71 62 37  min=o8hvl0th0qb7
0280: 64 36 6C 36 32 75 70 65 72 31 6E 67 63 6A 34 62  d6l62uper1ngcj4b
0290: 6F 75 33 6F 3B 20 70 6D 61 55 73 65 72 2D 31 3D  ou3o; pmaUser-1=
02a0: 67 75 44 62 4D 31 70 5A 61 51 30 25 33 44 0D 0A  guDbM1pZaQ0%3D..
02b0: 0D 0A D9 EA 22 1D 59 A8 A9 E7 20 2D F4 5A E7 D2  ....".Y... -.Z..
02c0: E2 CE 32 75 70 65 72 31 6E 67 63 6A 34 62 6F 75  ..2uper1ngcj4bou
02d0: 33 6F 26 70 68 70 4D 79 41 64 6D 69 6E 3D 6F 38  3o&phpMyAdmin=o8
02e0: 68 76 6C 30 74 68 30 71 62 37 64 36 6C 36 32 75  hvl0th0qb7d6l62u
02f0: 70 65 72 31 6E 67 63 6A 34 62 6F 75 33 6F 26 70  per1ngcj4bou3o&p
0300: 6D 61 5F 75 73 65 72 6E 61 6D 65 3D 72 6F 6F 74  ma_username=root
0310: 26 70 6D 61 5F 70 61 73 73 77 6F 72 64 3D 72 6F  &pma_password=ro
0320: 6F 74 31 26 73 65 72 76 65 72 3D 31 26 70 68 70  ot1&server=1&php
0330: 4D 79 41 64 6D 69 6E 3D 6F 38 68 76 6C 30 74 68  MyAdmin=o8hvl0th
0340: 30 71 62 37 64 36 6C 36 32 75 70 65 72 31 6E 67  0qb7d6l62uper1ng
0350: 63 6A 34 62 6F 75 33 6F 26 6C 61 6E 67 3D 65 6E  cj4bou3o&lang=en
0360: 26 63 6F 6C 6C 61 74 69 6F 6E 5F 63 6F 6E 6E 65  &collation_conne
0370: 63 74 69 6F 6E 3D 75 74 66 38 5F 67 65 6E 65 72  ction=utf8_gener
0380: 61 6C 5F 63 69 26 74 6F 6B 65 6E 3D 34 35 61 31  al_ci&token=45a1
0390: 62 38 33 39 35 31 31 34 30 32 32 64 66 31 37 39  b8395114022df179
03a0: 35 35 32 65 34 31 66 31 31 64 64 65 A4 11 E8 68  552e41f11dde...h
03b0: 45 7D DE ED 3F 74 81 1D 1D 9C A1 72 00 00 00 00  E}..?t.....r....
```

בכלליות, ניתן לראות שה-Dump כולל פרטי גלישה של אחד ממשתמש האתר, ובפרט:

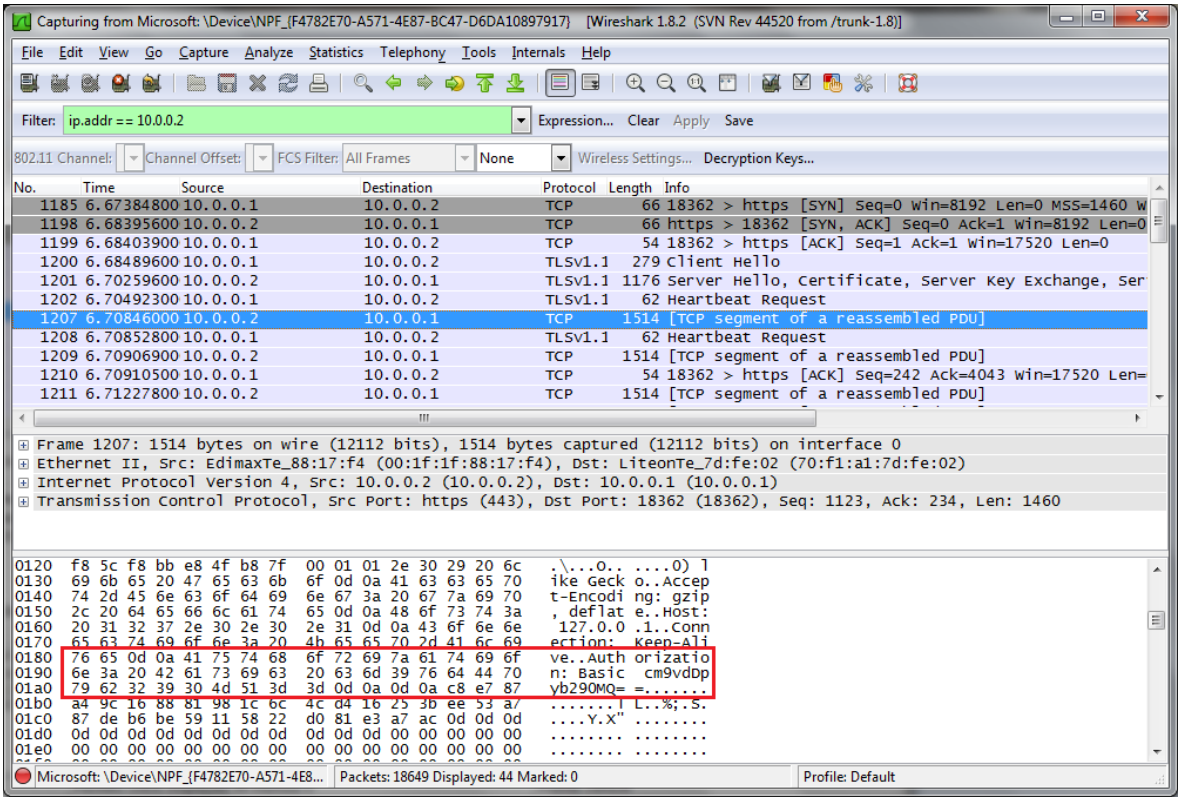
- **באדום** - ניתן לראות את פרטי שדה ה-Referrer של החבילה, הכולל את העמוד ממנו המשתמש הגיעה, בפרטים ניתן לראות את ה-Token שבו הוא משתמש.
- **בכחול** - ניתן לראות את פרטי העוגיה שבה המשתמש משתמש בעת הגלישה באתר.
- **בחום** - ניתן לראות את פרטי טופס ה-POST שהמשתמש שלח, בפרטים הנ"ל ניתן לראות את שם המשתמש (root) והסיסמה (root1) שהמשתמש הכניס על מנת לעבור את עמוד ה-Login.

מה שבאמת מעניין כאן הוא שהמידע הנ"ל נשלח ב-SSL, אך בתוך הזיכרון של התהליך הוא מפוענח ולכן אנו רואים את המידע כ-ClearText.

ב-PCAP הבאים ניתן לראות את אופן עבודת החולשה - תוקף שולח את חבילת ה-Heartbeat הזדונית:



ובמקום Heartbeat Response, מוחזרת החבילה הבאה (שגם לכריש קצת קשה לקבוע מה היא...):



כמובן שבדוגמא הנ"ל מדובר בגלישה שאני ביצעתי, אך זאת רק מפני שאני המשתמש היחיד באותו שרת, בעת מימוש החולשה - אין זה משנה מהיכן מריצים אותה, הפלט הנ"ל נזרק אלינו ישירות מה-Heap של התהליך, ושם אין הגבלה איזה מידע שייך לאיזו כתובת IP. בנוסף - איננו מוגבלים לכמות הפעמים שנריץ את המתקפה על השרת - במידה ומדובר בשרת רציני, עם משתמשים רבים - בכל הרצה נקבל מידע שונה, מה שמגדיל את הסיכויים לקבל מידע רגיש.

דוגמא נוספת אפשרית בעזרת מודול שנכתב ע"י לא מעט אנשים וניתן להוריד אותו מכאן:

- https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl_heartbleed.rb

דוגמא לשימוש:

- https://community.rapid7.com/community/metasploit/blog/2014/04/09/metasploits-heartbleed-scanner-module-cve-2014-0160

Reverse Heartbleed Attack

אם במתקפת Heartbleed התוקף הוא הלקוח, והנתקף הוא השרת, אז Reverse Heartbleed היא בדיוק אותו מימוש של מתקפת Heartbleed המקורית רק שכאן המתקפה מתבצעת ע"י שרת הנמצא בשליטתו של התוקף - והקורבן כאן הוא הגולש.

אם לקוח משתמש בתוכנת קליינט העושה שימוש ב-OpenSSL בגרסא פגיעה - השרת יוכל לנצל זאת על מנת לשלוף ממנו מידע. כאמור, החולשה קיימת במנגנון ה-Heartbeat, על פי ה-RFC, לא רק על השרת להגיב לחבילות אלו, אלא גם על הלקוח. מכאן שהשרת יוכל ליזום חבילות Heartbeat זדוניות על מנת לנסות ולדלות מידע על אתרים אחרים / משאבים אחרים בהם עושה הגולש בעת שימוש מקביל לגלישה אליו.

לדוגמא, אם אני גולש ל-Gmail שלי בעזרת דפדפן הפגיע למתקפה, ובמקביל אני גולש לכתובת URL שקיבלתי ממקור לא אמין, בעת הכניסה שלי לאותו שרת, השרת יכול לבקש מהדפדפן שלי ליזום תקשורת SSL (לדוגמא, ע"י טעינת קובץ מקישור HTTPS וכו') ולאחר מכן לשלוח לו חבילת Heartbeat זדונית ולנסות לדלות מידע אודות חיבור ה-SSL שלי לשרת ה-Gmail.

גם למקרה הזאת פורסמו כלים המאפשרים לבצע בדיקה כזאת, לדוגמא:

https://reverseheartbleed.com/

וכנ"ל מודול מקביל ל-Metasploit:

http://www.rapid7.com/db/modules/auxiliary/server/openssl_heartbeat_client_memory

התיקון

מאז פרסום המתקפה, בוצע תיקון לקוד ושוחררה גרסא המוגנת מפני המתקפה הנ"ל (1.0.1g), החלק הארי של התיקון הינו בתחילת הפונקציה. לחלק הנ"ל בקוד:

```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

הוסיפו Boundary Check באופן הבא:

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;
```

כך, במידה והחבילה ריקה, או אורכה שונה ממה שהמשתמש דיווח - מוחזר 0 והמשך הקוד לא מתבצע.

אז מה ניתן לעשות?

אז מה ניתן לעשות על מנת להתגונן מפני המתקפות הנ"ל? פשוט מאוד - לעדכן את גרסאות ה-OpenSSL שלכם לגרסאות שאינן פגיעות. ניתן להוריד את הגרסא האחרונה של OpenSSL מכאן:

http://www.openssl.org/source/

מומלץ מאוד להריץ את אחד הכלים שהוזכרו במאמר / קיימים באינטרנט לטובת גילוי הפגיעות ועדכון השירות שנמצא בפגיע.

אחת הבעיות העיקריות היא שבגלל שה-Information Disclosure הנ"ל נובע מחבילת Heartbeat ולא מבקשת GET / POST ודומותיהן, אין שום רישום בלוגים של שרתי האפליקציה (כדוגמאת קבצי לוג ה-Access / Error של שרתי ה-HTTP) וקשה מאוד יהיה לחסום חבילות אלו ברמת Firewall או WAF. ה-FBI פרסם חתימות Snort לטובת איתור וחסימת המתקפה:

חתימה גרסאת SSLv3:

```
alert tcp any any <> any
[443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](content:"|18 03 00|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1; msg: "SSLv3 Malicious Heartbleed RequestV2"; sid: 1;)
```

חתימה גרסאת TLSv1:

```
alert tcp any any <> any
[443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](content:"|18 03 01|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1; msg: "TLSv1 Malicious Heartbleed RequestV2"; sid: 2;)
```

חתימה גרסאת TLSv1.1:

```
alert tcp any any <> any
[443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](content:"|18 03 02|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1; msg: "TLSv1.1 Malicious Heartbleed RequestV2"; sid: 3;)
```

חתימה גרסאת TLSv1.2:

```
alert tcp any any <> any
[443,465,563,636,695,898,989,990,992,993,994,995,2083,2087,2096,2484,8443,8883,9091](content:"|18 03 03|"; depth: 3; content:"|01|"; distance: 2; within: 1;content:!"|00|"; within: 1; msg: "TLSv1.2Malicious Heartbleed RequestV2"; sid: 4;)
```

חשוב לזכור שלא רק שירותי HTTP פגיעים, גם שירותי IMAP, SMTP העושים שימוש במימוש פגיע של OpenSSL יכולים להיות פגיעים באותה המידה ובאותה החומרה. וכמובן - חשוב מאוד לא להכנס לקישורים שאיננו בטוחים במאה אחוז במקורם.

כמה באמת המתקפה הנ"ל פרקטית?

שלא כמו במתקפות הקודמות על משפחת הפרוטוקולים הנ"ל, המתקפה הנ"ל אינה מתקפה קריפטוגרפית, אלא מתקפת Information Disclosure שאינה מורכבת יחסית, ולכן קל מאוד לבצע אותה. מתקפות כמו BEAST או CRIME הן מתקפות קריפטוגרפיות, ולכן קשה באמת להעריך עד כמה ניתן יהיה לבצע אותן מחוץ למעבדה. לעומת זאת, המתקפה הנ"ל לא תלויה בתזמונים או בסטטיסטיקה.

כבר בעת פרסום ה-PoC שלה, הוצג כיצד ניתן לשלוף מידע פרטי מ-Yahoo, ובעת פרסום פרטי המתקפה, האתרים הנ"ל (ועוד רבים) היו פגיעים למתקפה:

- Facebook
- Instagram
- Pinterest
- Tumblr

- Twitter
- Google
- Yahoo
- Gmail
- Yahoo Mail
- GoDaddy
- Intuit Turbo Tax
- Dropbox
- Minecraft
- OkCupid

בקישור הבא ניתן לראות רשימה כוללת יותר של אתרים וסטטוס הפגיעות שלהם למתקפה:
http://mashable.com/2014/04/09/heartbleed-bug-websites-affected/

מקרה נוסף שדווח, פורסם כבר ב-16 לחודש. פורסם כי צעיר בן 19 בשם Stephen Arthuro מאונטריו, נעצר בביתו בשל החשד כי פרץ ל-Canada Revenue Agency וגנב כ-900 מספרי ביטוח לאומי.

ממחקר נוסף שבוצע ע"י חברת Mandiant, התגלה כי תשתיות VPN הן אחד היעדיים המרכזיים לחולשה זו, וכי **כבר ביום שלמחרת פרסום החולשה**, נתקפו מספר שרתי VPN המשמשים כשער-כניסה לרשתות של ארגונים שונים, ונגנבו מהם מפתחות פרטיים / פרטי הזדהות של סשנים פעילים ובהם נעשה שימוש על מנת להכנס לתוך אותן רשתות.

על מנת לבחון את הנושא, להגביר את המודעות, וכמובן - לקבל קצת פרסום, חברת Cloudflare פרסמה אתגר בנושא, הם הקימו שרת nginx-1.5.13 המשתמש ב-OpenSSL בגרסא פגיעה שהורץ על מערכת ההפעלה Ubuntu 13.10 וביקשו מכל מי שהיה מעוניין, להשיג את המפתחות הפרטיים של השרת, לחתום בעזרתם את המחרוזת:

```
Proof I have your key
```

ולשלוח להם אותו במייל.

הפותר הראשון של האתגר היה בחור בשם Fedor Indutny, אשר שלח **מעל 2.5 מיליון חבילות Heartbeat זדוניות** על מנת להשיג מספיק מידע שבעזרתו יוכל להרכיב את המפתחות הפרטיים של השרת.

ביבילוגרפיה וקישורים מומלצים לקריאה נוספת

האינטרנט מלא בקישורים ובמידע מעניין בנוגע למתקפה - ובעתיד הקרוב אני בטוח שיצוץ עוד מידע מעניין בנושא, ולכן אני ממליץ מאוד להשאר מעודכן, להלן מספר קישורים רלוונטים ומעניינים בנושא:

- http://heartbleed.com/
- https://tools.ietf.org/html/rfc6520
- http://blog.cryptographyengineering.com/2014/04/attack-of-week-openssl-heartbleed.html
- http://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=4817504
- http://blog.existentialize.com/diagnosis-of-the-openssl-heartbleed-bug.html
- http://vimeo.com/91425662
- http://www.msuiche.net/2014/04/10/eyebleed-a-technical-analysis-of-the-fix-not-the-bug-for-the-heartbleed-issue/
- https://www.us-cert.gov/ncas/alerts/TA14-098A
- https://community.rapid7.com/community/metasploit/blog/2014/04/09/metasploits-heartbleed-scanner-module-cve-2014-0160
- http://vrt-blog.snort.org/2014/04/heartbleed-memory-disclosure-upgrade.html
- http://blog.didierstevens.com/2014/04/09/heartbleed-packet-capture/
- http://www.garage4hackers.com/entry.php?b=2551
- http://blog.ioactive.com/2014/04/bleeding-hearts.html
- http://www.computersnyou.com/3155/2014/04/update-install-openssl-source-latest-version/
- http://wp.libpf.com/?p=535
- http://indy.fulgan.com/SSL/
- http://www.pcworld.com/article/2142808/reverse-heartbleed-puts-your-pc-and-the-internet-of-things-at-risk.html